VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Katedra informatiky

Modul pro statistická měření do programu UniSave

Statistic mesurment module for program Unisave

2012 **Jakub Fidler**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jakub Fidler**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Modul pro statistická měření do programu UniSave
Statistic Mesurment Module for Program Unisave

Zásady pro vypracování:

Cílem práce je rozšířit stávající program UniSave. Program UniSave je implementován v jazyce Java a slouží pro načítání hodnot z různých měřidel. Tyto naměřené hodnoty se dále zpracovávají a vytváří se z nich protokol o měření.

Rozšířená verze programu UniSave bude obsahovat následující:
1. Opakované měření jedné veličiny a její statistická vyhodnocení.
2. Zobrazení grafu naměřených hodnot.
3. Zobrazení histogramu naměřených hodnot.
4. Vytváření protokolu o měření.
5. Změna práce s XML soubory pomocí JAXB.

Modul pro podporu statistických měření do programu UniSave bude obsahovat:
1. Implementaci modulu pro statistická měření jedné veličiny (průměrná hodnota, směrodatná odchylka, minimum, maximum).
2. UML diagramy popisující danou implementaci.
3. Popis a dokumentaci implementovaného modulu.

Seznam doporučené odborné literatury:

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (Gang of Four): Návrh programů pomocí vzorů. Grada. Praha 2003. ISBN 8024703025

Brett McLaughlin: Java and XML, Second Edition. O'Reilly Media, August 2001. ISBN:978-0-596-00197-1

Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. David Ježek, Ph.D.**

Datum zadání: 19.11.2010

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 3. května 2012 ……

Poděkování

Chtěl bych velmi poděkovat svému vedoucímu práce Ing. Davidu Ježkovi, Ph.D., za jeho čas strávený u konzultací a poskytnutí důležitých rad, které mi pomohly při tvorbě bakalářské práce.
Dále děkuji své rodině za vytvoření ideálních studijních podmínek nejen při psaní bakalářské práce, ale také po celou dobu studia na vysoké škole.

Abstrakt

Cílem této bakalářské práce je rozšířit stávající program Unisave o opakované měření jedné veličiny a její vyhodnocení tzv. statistické měření. Aplikace slouží k načítání hodnot z různých měřidel. Na základě takto naměřených hodnot bude program Unisave schopen tyto veličiny vyhodnotit, vyhledá maxima, minima, vypočítá aritmetický průměr naměřených hodnot a směrodatnou odchylku. Následně tyto hodnoty vykreslí do grafu a histogramu. Celé měření se vygeneruje do závěrečného protokolu, který je možno uložit do mnoha různých formátů. V tomto modulu bude také upravená práce s ukládáním a načítáním dat, kdy místo technologie SAX bude použita technologie JAXB.

Klíčová slova:

Unisave, Java, GUI, Xml, JaxB

Abstract

The aim of this Bachelor Thesis is to supplement a current program Unisave, that is used for loading values from various gauges, with repeated measurements of one value and its evaluation, called static measurement. The program Unisave will be able to evaluate these values wheter it finds maximum, minimum, arithmetic mean of the measured values or standard deviation based on the measured values. Subsequently, it will draw a graph and histogram with these values. Work with the storage and loading of the data will be also modified in this module, technology JAXB will be used instead of technology SAX.

Keywords:

Unisave, Java, GUI, Xml, JaxB

Seznam použitých symbolů a zkratek

- DOM – Document Object Model
- SAX – Simple API for XML
- JAXB - Java Architecture for XML Binding
- XHTML –eXtensible HyperText Markup Language
- HTML – HyperText Markup Language
- GUI – Graphical User Interface
- XML – eXtensible Markup Language
- W3C- World Wide Web Consortium

Seznam obrázků

Seznam zdrojových kódů

OBSAH

PŘÍLOHY

1. Úvod

Program UNISAVE je implementován v jazyce Java. Používá se pro vyhodnocení naměřených dat z různých druhů měřidel. Tyto měřidla se dají připojit k počítači pomocí sériového, popřípadě infračerveného portu. Nebo je také možno data vkládat ručně. Prozatím program Unisave podporoval pouze XY měření, které snímá data ze dvou měřidel a následně je vyhodnocuje.

Cílem této práce je naimplementovat nový druh měření jedné hodnoty, takzvané statistické měření. Princip spočívá v tom, že pomocí měřidla se měří například tloušťka nátěru na různých místech požadovaného objektu. Z měřidel nám mohou chodit naměřené data v různých jednotkách, proto je zde naimplementován převod jednotek, který umožňuje jednotky mezi sebou převádět, pokud je to možné. Po na měření, se data vyhodnotí a zobrazí v grafu popřípadě, v histogramu naměřených hodnot. Dále se z naměřených dat vypočítají statistické údaje, jako je směrodatná odchylka, minimální hodnota, maximální hodnota a aritmetický průměr. Všechny tyto zpracované informace jsou přeneseny do protokolu o měření.

Následující kapitoly popisují postup při tvoření statistického modu a změnu ukládání a načítání dat. První část této práce je věnovaná právě změně práce se soubory. Zjednodušeně je zde vysvětleno, co je to jazyk XML a jak vypadá jeho struktura. Dále se budu věnovat způsobu spolupráce Javy s XML technologiemi a porovnáním jejich vlastností. Poté následuje popis samotné implementace, a způsob řešení problémů, se kterými jsem se při této práci setkal.

Druhá část je věnovaná vytváření samotného modulu pro statistické měření. Jsou zde popisovány jednotlivé třídy a jejich úpravy a implementované metody, které jsou následně zobrazeny v sekvenčních diagramech. Větší pozornost bude také věnována vytváření objektů třídy *Action*, která se v programu Unisave využívá pro tvorbu nabídkového menu a nástrojové lišty. Díky rozdělení grafické části od části datové, je možné implementaci rozdělit na tvorbu grafického rozhraní a tvorbu datové části.

2. Ukládání dat

Ukládání a načítání nastavení je nedílnou součástí programu Unisave. Řeší se pomocí práce s XML. XML(eXtensible Markup Language) je na systému nezávislý jazyk, který slouží pro vyjadřování dat a jejich struktury. Dokument XML je textový soubor Unicode znaků, který obsahuje data spolu se značkami, které definují strukturu dat. Jelikož je XML textovým souborem, je možné ho vytvořit v jakémkoliv textovém editoru. Původ XML se objevuje v referenčních příručkách pro stroje a programy od firmy IBM z 60. let minulého století [1]. Respektive je to zvláštní podmnožina jazyka SGML, jenž je univerzální značkovací metajazyk, který umožňuje definovat značkovací jazyky jako své vlastní podmnožiny. SGML je komplexní jazyk poskytující mnoho značkovacích syntaxí, ale jeho složitost brání většímu rozšíření [5].

Značkování XML a HTML může vypadat podobně, protože se u obou používají tagy a atributy, které jsou poté ukládány do textového souboru. Toto je jediná podobnost mezi oběma jazyky. Důležitější je zásadní rozdíl mezi účelností a schopností těchto dvou jazyků. Jelikož HTML porušovalo některé zásady XML, bylo vytvořeno takzvané XHTML, které tyto přestupky řeší. Jako příklad si uvedeme zákaz křížení atributů nebo pravidlo, podle kterého musejí být všechny hodnoty v uvozovkách [3]. Nicméně vývoj XHTML byl ukončen.

Zatím co HTML jazyk má omezený počet tagů a je určen především pro popis zobrazení dat (HTML je určen pro prezentaci dat), tak XML je určeno spíše pro přenos informací, které budou zpracovávat počítačové programy.

2.1 STRUKTURA XML

Dokument XML se skládá ze dvou částí (viz obr. 1):

PROLOG (hlavička) - uvádí informace nezbytné pro interpretaci obsahu těla dokumentu. Skládá se ze dvou volitelných částí a to deklarací XML a deklarací typu dokumentu.

TĚLO DOKUMETU - obsahuje data, skládá se z jednoho nebo několika elementů. Každý element je definován počátečním a koncovým tagem.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<výrobek>
    <název>Kolo</název>
    <cena>356</cena>
    <měna>$</měna>
</výrobek>
```

Obrázek 1: Ukázka Xml

2.2 XML a Java

Java obsahuje několik API pro práci s XML

1, SAX (Simle API for XML)

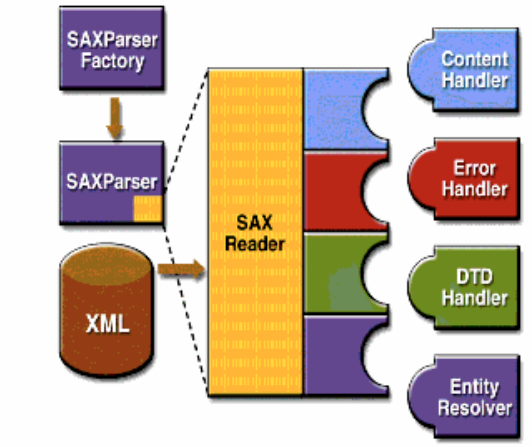
2, DOM (Document object model)

3, JAXB (Java Architecture for XML Binding)

2.2.1 Porovnání Java technologii

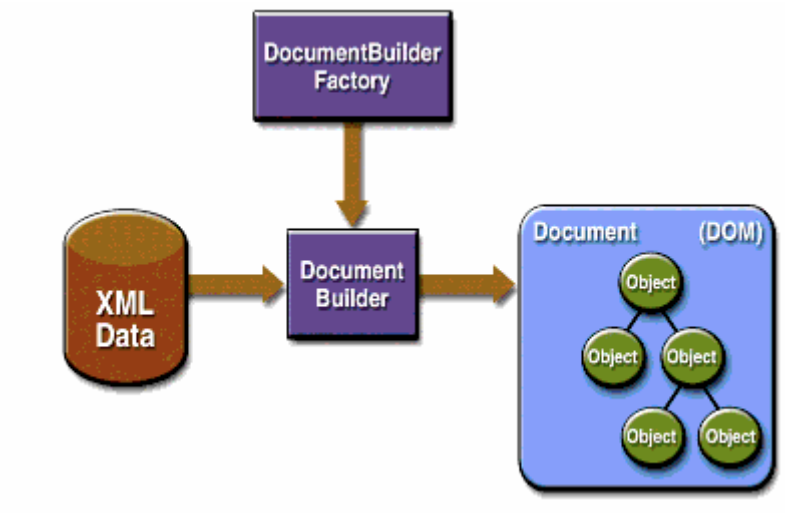
V Jave je možno využít SAX nebo DOM parser pomocí JAXP API, a tím analyzovat XML dokument. Což znamená, načíst celý soubor a logicky jej rozdělit na jednotlivé kusy. Rozparsrovaný dokument je pak k dispozici aplikaci.

SAX parser začíná na začátku dokumentu a každá oddělená událost, jako je začátek nebo konec elementu, vyvolá určité metody, které jsou spojeny s touto událostí (*handler*). SAX nemá přehled o tom, jak vypadá celý dokument, proto je potřeba, aby si ho programátor sestavil pomocí informací z implementovaných metod. Tím, že u SAX není potřeba načítat celý dokument, je tato technologie velmi rychlá a nenáročná na paměť (obr. 2).



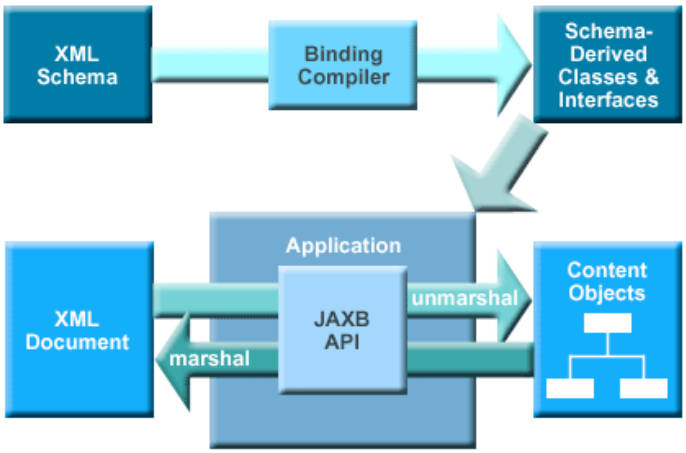
Obrázek 2: Model Sax

Naopak u technologie DOM je celá struktura sestavena v paměti a vrácena jako objekt typu *Document* (obr. 3). Z něj lze pak volat jednotlivé metody tohoto objektu a procházet jednotlivými elementy. Velkou výhodou je, že tento dokument je k dispozici tolikrát a takovou dobu, po kterou ho programátor vyžaduje. Naproti tomu nevýhodou je, že je celý uložen v paměti, což může být u velkých XML souborů problém. Na rozdíl od SAXU dokáže editovat existující XML soubory nebo vytvářet nové.



Obrázek 3: Model Dom

JAXB převádí Java objekty na XML a naopak (obr. 4). Umožňuje Java programátorům přistupovat a zpracovávat data XML, aniž by museli znát XML nebo zpracování XML. Na rozdíl od metody SAX zde není potřeba vytvářet SAX parser nebo psát metody zpětného volání. Pokud nám nějaká třída generuje SAX události, lze tyto události poslat do JAXB, které z nich sestaví data v potřebné struktuře. Také umožňuje přístup k datům v náhodném pořadí, a na rozdíl od DOMU nás nenutí procházet celý strom pro přístup k datům. JAXB může načítat data z různých vstupních zdrojů od souboru, potomka třídy *ImputStream*, URL adresy, DOM uzlu nebo transformovaných dat objektu. Objekty ve stromu vytvořené pomocí JAXB jsou efektivnější na paměť než u technologie DOM.



Obrázek 4: Model JAxB

Schéma je XML specifikace, která identifikuje prvky v dokumentu XML, určuje, v jakém pořadí mají být uvedeny, které atributy se zde mohou vyskytovat. XML vytvořené pomocí JAXB nevyžaduje schéma, ale pokud se s ním bude pracovat, je potřeba, aby dané schéma bylo napsané pomocí standartu W3C XML schema.

2.3 Převod Java objektu na XML

Pro správnou práci s JAXB budeme potřebovat naimportovat balíček *javax.xml.bind*. V tomto balíčku jsou implementovány třídy a rozhraní pro provádění operací, jako je *unmarsahllig, marshallig*, validace apod. Funkcí *u*nm*arsahllig* se rozumí převod XML na Java objekty. A naopak převádění Java objektů na XML se nazývá *marshalling*.

Aby se převedly Java objekty do XML, je potřeba mít ve třídě implementovaný bezparametrický konstruktor, a na celou třídu přidat anotaci pomocí *@XmlRootElement* (označuje, že objekty dané třídy se budou serializovat).

Máme třídu `TestTest`, ve které se nacházejí proměnné (viz kód 1). Pro samotnou serializaci je potřeba si vytvořit *JAXBContect*. Jako parametr se udává typ nebo řetězec se jmény balíčků. Vytvořením nové instance získá JAXB cestu pro kompilátor. Dále je potřeba vytvořit samotný *marsahaller*, který řeší samotný převod. Pomocí *setProperty* můžeme nastavit například formátování. Poté již probíhá samotný převod (viz kód 2).

```
@XmlRootElement
public class TestTest {

    public String jmeno ="Jan";

    public String prijmeni  = "Sobota";

    public int vek = 24;

}
```

Kód 1: Třída TestTest

```
JAXBContext context =JAXBContext.newInstance(TestTest.class);
Marshaller marshaller = context.createMarshaller();
TestTest t = new TestTest();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
marshaller.marshal(t,System.out);
```

Kód 2: Převod Java objektů na xml

Výsledkem takto převedené třídy do XML je výpis, kdy RootElement je pojmenován po názvu třídy a elementy podle názvu proměnných (viz obr. 5).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<testTest>
    <jmeno>Jan</jmeno>
    <prijmeni>Sotbota</prijmeni>
    <vek>24</vek>
</testTest>
```

Obrázek 5: Převedené Java objekty na Xml pomocí JaxB

2.3. Převod XML souboru na Java objekty pomocí JAXB

Převod XML funguje na principu vytvoření stromu objektů. Nikoliv však na bázi jako DOM. Princip převodu je podobný jako při převádění Java objektů na XML (viz kód 3).

```
JAXBContext context = JAXBContext.newInstance(TestTest.class);
Unmarshaller unmarsahller = context.createUnmarshaller();
TestTest t = new TestTest();
t= (TestTest)unmarsahller.unmarshal(new InputStreamReader
        (new FileInputStream(f),"UTF-8"));
```

Kód 3: Převod xml na Java objekty

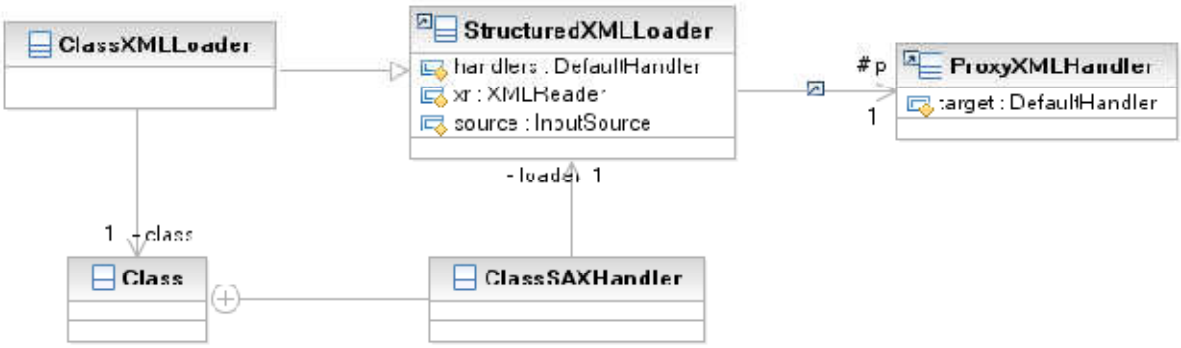
2.4 Soubory potřebné pro běh programu UNISAVE

V balíčku *resource* se nachází sedm XML souborů. Tyto dokumenty jsou jednotlivě načítány v průběhu běhu programu. Při startu programu se načítají data z *Unit.xml,* který obsahuje informace o měrných jednotkách (název, zkratka) a o jejich dostupnosti prefixů respektive názvech prefixů, jejich zkratkách a exponentech. Dále jsou zde popisovány konverze mezi jednotlivými jednotkami. Následným krokem je načtení možných měřících zařízení, která jsou popsány v *mesurmentDevice.xml*. V souboru jsou přesně definovány názvy měřících přístrojů, jejich nastavení, množina použitelných měrných jednotek a další specifické vlastnosti měřidel. Jedním z dalších načtení, při startu programu je *Unisave2006_UserSetting.xml*. Zde jsou popisována uživatelská nastavení, jako například počet dostupných portů, naposledy otevřený soubor, domovský adresář nebo naposledy použitá jednotka. Posledním základním načteným souborem je *unisavesetting.xml*, ve kterém je pospána cesta k šabloně pro vytvoření protokolu. Udává počet naposledy otevřených měření a název aplikace umožňující komunikaci mezi programem Unisave a fyzickými měřidly. Abychom mohli vytvořit protokol o měření, je v balíčku *resourse* také *reportXY.xml*, který slouží jako šablona protokolu.

2.5 Původní stav

V původní verzi programu UNISAVE byla použita technologie SAX. Výchozí technologii SAX můžeme popsat pomocí obecného digramu zobrazeného na obrázku č. 6. Základní třídou je třída `StructuredXMLLoader`, ve které je řešena veškerá logika načítání dat. `ProxyXMLHandler` atribut určuje, kdo bude aktuálně zpracovávat načtený soubor. Třída *`Class`* je reprezentována v programu UNISAVE především jako `GlobalSetting` a `UserSetting` v balíčku *setting*, jenž dědí ze třídy `ClassXMLLoader` (`DeviceXMLLoader`, `MesurmentXMLLoader` atd.). Tato třída obsahuje metody *StartElement() a endElement()* reagující na začátky a konce tagů v XML. [4]

`ClassXMLLoader` vytvoří nebo předá odkaz na třídu, která se má načítat (`Class`). Tato třída obsahuje přidanou třídu `ClassSaxHandler`, která velmi zjednodušuje inicializaci, kontrolu a manipulaci s atributy tím, že třída má přímý přístup k privátním atributům. Ve chvíli, kdy dojde ke špatnému načtení souboru, udržuje `ClassSaxHandler` pomocí atributu *loader* odkaz na předcházející parser. `ClassSaxHandler` je možné vynechat v případě načítání malého objemu dat.



Obrázek 6: UML diagram původního načítání

2.6 Změna technologie ukládání v programu Unisave

Celá práce s načítáním a ukládáním dokumentů je řešena v balíčku *unisave2006.settings*. V tomto balíčku se nacházejí třídy *`GlobalSetting`* a *`UserSetting`*, které jsou základním stavebním kamenem pro práci s XML. V obou těchto třídách jsou implementovány metody pro inicializaci jednotlivých částí programu, které jsou nezbytné pro jeho chod. Ať už se jedná o uživatelské nastavení, načítaní měřících zařízení či měrných jednotek.

2.6.1 Změna načítání Usersetting

Inicializace uživatelského nastavení programu je implementována v třídě `UserSetting`, ve které jsou deklarovány všechny proměnné. JAXB se orientuje pomocí anotací, takže bylo potřeba nastavit anotaci na celou třídu jako *@XmlRootElement* a *@XmlAccessorType(XmlAccessType.PROPERTY),* která určuje serializovatelnost. Tím, že zde není implementovaný žádný parser, JAXB automaticky serializuje všechny proměnné a property v této třídě. Což je nežádoucí, neboť některé informace se již nacházejí v jiném XML, tak je není třeba znovu ukládat. Proto bylo potřeba na některé proměnné, get a set metody použít anotaci *@XmlTransient*, která dané proměnné učiní neviditelnými pro JAXB. Tato anotace se může hodit také v případě, že se JAXB zanoří příliš hluboko do objektu a serializuje pro nás zbytečně podrobné informace o objektech, které si nežádáme. V tomto případě se jedná o proměnné *lastUsetDevice* a *lastUsetSecondDevice*. Bylo tedy nutné opět tyto proměnné skrýt a zároveň vytvořit speciální get a set metody pro námi zvolené proměnné. Konkrétně se jednalo o názvy měřidel (viz kód 4). Byla potřeba také upravit třídy, s kterými třída `UserSetting` pracuje (*PortNames, Units, ProtokolSetting, XYGraphSetting*). Především jde o přidání dalších anotací na proměnné *(@XmlElement* popřípadě *@XmlAttribute*), třídy *(@XmlRootElement*) nebo vytvoření prázdného konstruktoru nezbytného pro serializaci pomocí JAXB. Další problém byl se serializováním objektu typu `UnitInterface`, kdy JAXB neumí pracovat s rozhraním. Tomuto tématu se budu více věnovat později u načítání měřidel.

```
@XmlTransient
      protected MesurmentDevice lastUsetDevice = null;

   @XmlTransient
      public MesurmentDevice getLastUsetDevice() {
            return lastUsetDevice;
      }


   public String getLasetUsetDeviceName() {
        return lastUsetDevice.getName();
    }

    public void setLasetUsetDeviceName(String lasetUsetDeviceName) {
        for(MesurmentDevice md :
               GlobalSetting.getInstance().getMesurmentDeviceSet()){
            if(md.getName().equals(lasetUsetDeviceName)){
                lastUsetDeviceN = md;
            }
        }
    }
```

Kód 4: Úpravy Usersetting

Inicializace `UserSetting` původně probíhala tak, že byl vytvořen *UserSettingXMLLoader*, ve kterém byl implementovaný samotný SAX parser. Při použití JAXB nám použití třídy *`UserSettingXMLLoader`* odpadá a pomocí metody *unmarsahller* se převede XML na Java objekty (viz kód 5).

```
/*
* inicializacia uzivatelskych nastaveni z xml suboru
*/
private static void initializeUserSetting() {
    JAXBContext context;
    File setting = new File("C:/UniSave2006_UserSetting.xml");
      try {
           context = JAXBContext.newInstance(UserSetting.class);
           Unmarshaller unmarsahller = context.createUnmarshaller();

            try {
                userSetting =(UserSetting) unmarsahller.unmarshal(
                              new InputStreamReader (new
                              FileInputStream(setting), "UTF-8"));

                } catch  UnsupportedEncodingException e) {
                       // TODO Auto-generated catch block
                       e.printStackTrace();


      } catch (JAXBException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
      }
```

Kód 5: Inicializace UserSetting

Unisave2006_UserSettings.xml je jediný soubor, který se načítá a zároveň ukládá. Ukládání je řešeno v metodě *store(),* kdy naopak byl použit *marsahller*, který serializuje třídu `UserSettings` do XML. Důležité je u výstupu správně nastavit kódování UTF-8 pro dobrou čitelnost znaků.

2.6.2 Změna načítání MesurmentDevice

Volání metody pro načítání jednotlivých měřidel je implementováno v třídě `GlobalSetting`. Přesněji v metodě *initializeMesurmentDevice()*, kde je vytvořený další *JAXBcontext* s instancí na třídu `MesuementDeviceSet`. Tato třída obsahuje seznam všech měřidel a nastavení, potřebné pro načítání dat z měřidel. Podobně jako u `Usersetting` bylo nezbytné anotovat jednotlivé třídy podle potřeby *(@XmlElement, @XmlRootElement*, popřípadě *@XmlTransient*). Největším problémem při načítání měřidel bylo načtení *commDeviceSetting*, který je implementován pomocí rozhraní s kterým si JAXB neumí poradit. Bylo tedy zapotřebí přidat anotace na interface *@XmlTransient* a *@XmlSeeAlso ({CommDeviceSetting.class ,NullComDeviceSetting.class}). @XmlSeeAlso* anotace nám říká, že

z rozhraní má JAXB přejít na třídy *CommDeviceSetting* nebo *NullComDeviceSetting*. A tyto třídy jsou opět upravené podle potřeby anotacemi. Stejně je to řešeno u `GrabberInterface`, který anotací odkazuje na `AbstractGrabber`. Odtud pak *@XmlSeeAlso* odkazuje na jednotlivé druhy měřidel *({MitutoyoAsciiGrabber.class, MahrAsciiGrabber.class, ManualGrabber.class, AsciiGrabber.class ,ElektroPhysicAsciiGrabber.class})*. Další úpravou ve třídě `AbstractGrabber` je vytvoření set metody (viz kód 6), která nám pomocí id jednotky načte dostupné jednotky pro dané měřidlo. Ta samá metoda musela být vytvořena taktéž v `ManualGrabber`, protože jako jediná dědí přímo z `GrabberInterface` a nikoliv z `AbstractGrabber`.

```
public void setPosibleUnits(Collection<UnitDescription> col){
  Map<Integer,UnitDescription> mainSet = GlobalSetting.getInstance().
                                        getUnitSet().getUnitDescriptions();
  posibleUnits = new Vector<UnitDescription>();

  for(UnitDescription item : col){

          posibleUnits.add((UnitDescription)mainSet.get(item.getId()));

  }
}
```

Kód 6: Změna metody setPosibleUnits

2.6 .3 Změna načítání Units

Podobně jak u `MesurmentDevice`, je načítání jednotek voláno ve třídě `GlobalSetting` v balíčku *unisave2006.settings*. Protože se původně nedařilo pracovat ve třídě `UnitSet` a `PrefixSet` s mapami, byla vytvořena speciální třída `unitDescriptionSet` v balíčku *units*, ve které byly vytvořeny tři proměnné typu *ArrayList*. Pro konverzi, jednotky a prefixy (viz kód 7). Dále se implementovaly metody *UnitSet getUnitSet()* a *PrefixSet getPrefixSet()* (viz kód 8), které jsou poté následně volány v `GlobalSetting` po metodě *unmarshall*.

```
@XmlElementWrapper(name="prefixes")
@XmlElement
public ArrayList<UnitPrefix> pref= new ArrayList<UnitPrefix>();
```

Kód 7: Ukázka ArrayList

```
UnitSet getUnitSet(){
```

```
        UnitSet us = new UnitSet();
        for(UnitDescription unitD : unit){
              for(UnitConversion uncon : conv){
                    if(unitD.getId() == uncon.getFrom().getId())
                          unitD.addConversion(uncon);
                                      }
              us.add(unitD);
        }

        return us;
}
PrefixSet getPrefixSet(){
        PrefixSet ps = new PrefixSet();

        for(UnitPrefix prefD : pref){
              ps.addPrefix(prefD);
}
        return ps;
```

Kód 8: Metody getUnitSet a getPrefixSet

Potřeba bylo taktéž upravit samotné třídy reprezentující prefixy (*UnitPrefix*) a konverze (*UnitConversion*), kde se pro lepší přehlednost místo anotací *@XmlElement* použili *@XmlAttribute*. Vytvořil se prázdný konstruktor. A u jednotek (*UnitDescription*) bylo provedeno to samé, navíc byla upravena metoda *addConversion(UnitConversion uc)*, která původně porovnávala objekty. JAXB však vytvářel jiné objekty, proto bylo zapotřebí implementovat porovnávání pomocí *id* (viz kód 9).

```
public void addConversion(UnitConversion uc) {
        if (uc != null && this.id == uc.getFrom().id)
              /*
               * getTo().getId() - znamena na co sa da skonvertovat je
                  kluc hasha
               */
              conversions.put(uc.getTo().getId(), uc);
}
```

Kód 9: Úprava metody addConversion

2.6.4 Změna načítání Unisave_setting

Posledním načítaným souborem je *unisave_setting*, který se volá v třídě *GlobalSetting*, ale je implementován ve vnořené třídě *GlobalSettingXMLLoader* v metodě *jaxbload()*. Jako jediný

soubor není načítán hned při startu programu, ale až v jeho průběhu. Takže by docházelo k přepisování objektu *globalSettings*. Proto byl vytvořen nový objekt *globalSettings2*, pomocí kterého se nastavily požadované proměnné bez negativních vlivů na ostatní proměnné (viz kód 10).

```
globalSettings2 =(GlobalSetting) unmarsahller.unmarshal(new
InputStreamReader(new FileInputStream(global), "UTF-8"));

reportXYFileName =globalSettings2.getReportXYFileName();
commBridgeExeFile=globalSettings2.getCommBridgeExeFile();
recentFileMaxCount=globalSettings2.getRecentFileMaxCount();
```

Kód 10: Načtení unisave_setting

2.6.5 Ukládání a načítání měření

Aby program Unisave mohl načítat uložená měření, byla potřeba přepsat ve třídě `MainFrame` metodu *openMesurment()*. Velkou změnou oproti ostatním převádění Java objektů do XML bylo to, že třída `Mesurment` (obsahuje měření) je abstraktní, tudíž na ní nešla vytvořit instance. Bylo tedy nutné vytvořit instance na třídy, které z ní dědí (`XYMesurment` a `StatisticMesurment`) (viz kód 11). Pokud by se v budoucnu doimplementovával další modul, bude potřeba tyto instance rozšířit o další třídy reprezentující typ měření. Pro načítání starších souboru, nebyla technologie SAX smazána, ale přesunuta do části, která je vyvolána v případě, že nefunguje převod pomocí JAXB. Přesněji do části vyvolávající její výjimku.

Aby se jednotlivá měření mohla načítat, bylo potřeba tyto data nejdřív uložit. K tomu sloužila metoda *saveToXML()*ve třídě `Mesurment`. V této třídě kromě připsání anotací k proměnným (podobně jako v předešlých případech), bylo nutné vytvořit několik get a set metod. Jednalo se především o nastavení grafu, popřípadě proměnných *Xmaker* a *Ymaker*, které jsou důležité pro načítání měření typu XYMesurment.

```
context=JAXBContext.newInstance(XYMesurment.class,StatisticMesurment.class)
```

Kód 11: Vytvoření kontextu pro načítání měření

2.7 Závěr změn ukládání

Při změně technologie ze SAX na JAXB bylo zapotřebí nejprve vytvořit nové XML soubory s podobnou strukturou. Podle toho se pak k jednotlivým proměnným, popřípadě k get a set metodám

taky přidávaly anotace. Po úspěšném vytvoření XML souboru se implementovalo jeho načítání, při kterém se ověřilo správné načtení všech hodnot. Velký důraz byl kladen především na vzhled XML a na jeho přehlednost a čitelnost. V tomto je JAXB vynikající. Sám totiž dokáže odsazovat jednotlivé elementy, čímž dělá XML strukturované a přehledné.

3.Implementace statistického modulu

Celý program Unisave je strukturován do několika balíčků, které oddělují funkcionalitu od grafického zobrazení. Proto se následující kapitoly budou pohybovat mezi balíčkem gui a balíčkem data. V těchto dvou složkách se mimo jiné nacházejí předpřipravené třídy pro statistické měření, které je potřeba doimplementovat a dosadit do již dané funkční struktury.

3.1Vytvoření grafické podoby modulu pro statické měření

Grafická podoba programu Unisave obsahuje kromě panelů pro statistické měření, také nástrojovou lištu s mnoha tlačítky, které se také nacházejí v nabídkových položkách. Vytvoření těchto komponent je realizováno pomocí objektů *Action*.

3.1.1 Tvorba nástrojové lišty pomocí rozhraní Action

Akce je objekt jakékoliv třídy, jenž implementuje rozhraní *Action*. (třídu již implementuje *AbstractAction*). Toto rozhraní deklaruje metody, které pracují s nějakým akčním objektem. *Action* rozšiřuje *ActionLisener*, takže objekt typu *Action* je zároveň přijímačem a zároveň akcí.[1]

Velkou výhodou je, že některé komponenty jako například *JMenu* nebo *JTooBar*, mají metodu *add()*, která přijímá jako argument objekty typu *Action*. Když tedy k těmto komponentám připojíme daný objekt, metoda z tohoto objektu pak vytvoří komponentu správného typu. Pokud je do objektu *JToolBar* pomocí metody *add ()* přidán objekt typu *Action,* je vrácena položka typu *JButton*. To samé platí v případě *JMenu*, kdy bude vrácen objekt typu *JMenuItem*. Toto dynamické tvoření „správných objektů“ je výhodné, jestliže chceme stejný objekt *Action* vložit do různých komponent, například do nabídky nebo nástrojové lišty.

Action rozhraní

Toto rozhraní je uzpůsobeno k ukládání sedmi základních vlastností:

- Jméno – slouží jako nápis pro položku nabídky nebo tlačítka nástrojové lišty
- Malá ikona – zobrazí obrázek na nástrojové liště
- Krátký popis – používá se jako popiska
- Akcelerační klávesa – definuje se objektem *KeyStore*
- Dlouhý popis – využívá se jako kontextová nápověda
- Mnemotechnický klíč – kód klíče
- Příkazový klíč akce

Obsahuje také několik metod např.

putValue() - ukládá hodnotu s klíčem
setEneabled() - nastavuje se zda je objekt povolený či nikoliv. Funguje jak pro tlačítko z nabídkové lišty, tak z lišty nástrojové.

3.1.2 Implementace Action v progamu Unisave

Program Unisave obsahuje několik objektů *Action*, proto na popis implementace bude vybrán pouze jeden a to objekt *ActionCopy*. Aby byla implementace přehledná, pracuje se v aplikaci s třemi třídami.

První třída je `AbstractActionCopy`, jenž dědí ze třídy `AbstractAction` (implementuje rozhraní *Action*), se nachází ve speciálním balíčku *gui.action*. V konstruktoru této třídy pomocí metody *putValue*() nastavujeme jednotlivé vlastnosti (viz kód 12).

```
public AbstractActionCopy() {
      super("Kopírovat");
      setEnabled(false);
      putValue(Action.SHORT_DESCRIPTION,
                  "Zkopíruje označené hodnoty do schránky.");
      putValue(Action.MNEMONIC_KEY, KeyEvent.VK_K);
      putValue(Action.ACCELERATOR_KEY,
                  KeyStroke.getKeyStroke("control C"));
      putValue(Action.ACTION_COMMAND_KEY, "CopyEntities");
      putValue(Action.DISPLAYED_MNEMONIC_INDEX_KEY, 0);
      putValue(Action.LARGE_ICON_KEY, new
                  ImageIcon(getClass().getResource(
                  "/resource/icons/copy.gif")));
      putValue(Action.LONG_DESCRIPTION,
                  "Zkopíruje vybrané hodnoty měření do schránky.");
      putValue(Action.SMALL_ICON, new
                  ImageIcon(getClass().getResource(
                  "/resource/icons/copy_small.gif")));
}
```

Kód 12: Konstruktor třídy AbstractActionCopy

Dále je zde vytvořena metoda *actionPerformed()*, která je prázdná, protože doposud není jisté, co se s akcí položky *ActionCopy* bude dít.

Druhá třída je vnořená třída ve třídě `StatistcMesurmentPanel` a jmenuje se `ActionCopy`. V konstruktoru této třídy přijímáme kostruktor z dědící třídy `AbstractActionCopy`. Dále se v této třídě nachází implementace události s názvem *actionPerformed()*, která volá funkci *getStatisticMesurmentValuePanel().copySelectedItems*(), jenž vloží vybrané záznamy do systémové schránky.

Ve třetí třídě `StatisticMesurmentPanel` pak dochází k samotnému vytvoření objektu *ActionCopy* v metodě *getActionCopy(),* která je následně volaná při inicializaci *MenuAction*. V této metodě můžeme vidět, že objekt typu *Action* nastavujeme jak pro *JButtonCopy,* tak pro *JMenuItemEditCopy*. (viz kód 13)

```
getJButtonCopy().setAction(panelInterface.getActionCopy());
getJMenuItemEditCopy().setAction(panelInterface.getActionCopy());
```

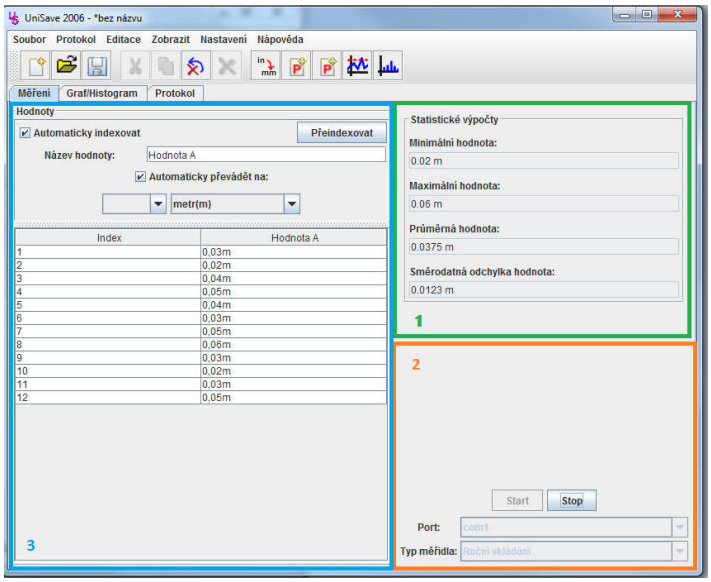
Kód 13: Připojení objektu Action

3.1.3 StatisticMesurmentPanel

Startovací třídou implementace modulu pro statické měření byla třída `StatisticMesurmentPanel` v balíčku *gui*. Zobrazí se po zvolení tlačítka nové měření. V této třídě se již nacházely předpřipravené metody, které vytvářely základní panely a byly připraveny pro přidání dalších grafických prvků. Celý `StatisticMesurmentPanel` obsahuje 3 záložky *JPanelMesurment, JPanelGraph ,JPanelProtocol*. *JPanelMesurment* obsahuje důležité prvky týkající se statistického měření (viz obr 7).

Jedná se především o tyto panely:

- JPanelMesurmentStatisticCalculation
- GraberPanel
- JPanelMesurment



Obrázek 7: StatisticMesurmentPanel

JPanelMesurmentStatisticCalculation

Panel byl vytvořen za účelem zobrazování statistických výpočtů (maxima, minima, průměru a směrodatné odchylky). Jednotlivé hodnoty jsou zobrazovány pomocí *JTexField*, a popsány

komponentou *Jlabel*. Každé textové pole má napsanou svoji get metodu, v které se vytváří. V metodě *refreshCalculation()* se pak pomocí metody *setText* tyto textová pole nastavují. Zároveň se připojují jednotky, v kterých jsou statistické výpočty počítány. Metody pro výpočet budou zmíněny později, když se budu zabývat popisem třídy *StatisticMesurment* (kapitola 3.2).

GraberPanel

Tento panel reprezentuje výběr měřidla a portu. Registruje se, jako posluchač na zpracování naměřených hodnot. V tomto panelu nebylo potřeba dělat jakékoliv úpravy, neboť již byl připojen k *JPanelMesurment.*

JPanelMesurment

Hlavním panelem v záložce *JPanelMesurment* je *StatisticMesurmentValuePanel*, který je reprezentován samostatnou třídou, proto se jí budu věnovat v další kapitole (kapitola 3.2.1).

Tyto tři panely jsou základním rozdělením záložky *JPanelMesurment*. Další záložkou je *JPanelGraph*. Ten obsahuje komponenty pro nastavení vlastností grafu, jako je nadpis a popis os, popřípadě orientace grafu. Opět je každý tento prvek vytvořen vlastní get metodou. V těchto metodách se většinou implementuje posluchač, který pak reaguje na nějakou událost (viz kód 12). V ukázkovém kódu se přidává posluchač a říká, že při změně textu v textovém poli se změní nadpis grafu.

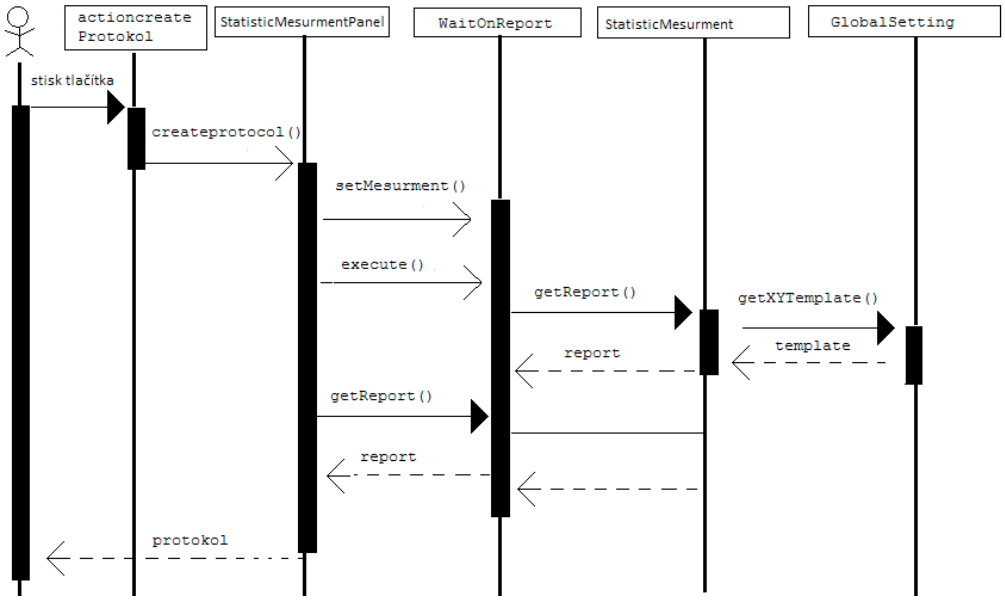
```
private JTextField getJTextFieldGraphTitle() {
      if (jTextFieldGraphTitle == null) {
          jTextFieldGraphTitle = new JTextField();
          jTextFieldGraphTitle
              .addFocusListener(new  java.awt.event.FocusAdapter() {
                   public void focusLost(java.awt.event.FocusEvent e) {
                       statMesurment.getGraphSetting().setGraphTitle(
                          getJTextFieldGraphTitle().getText());
                                  }
                              });
      }
      return jTextFieldGraphTitle;
}
```

Kód 14: Ukázka komponenty JTextField

Protokol je poslední záložkou, která se zobrazuje na tomto daném panelu. A obsahuje velké množství textových polí, které jsou však implementovány ve třídě *OrganizationPanel*. Pro nastavení datumu měření je zde použita třída `DateSelector`.

Metoda pro vytvoření protokolu

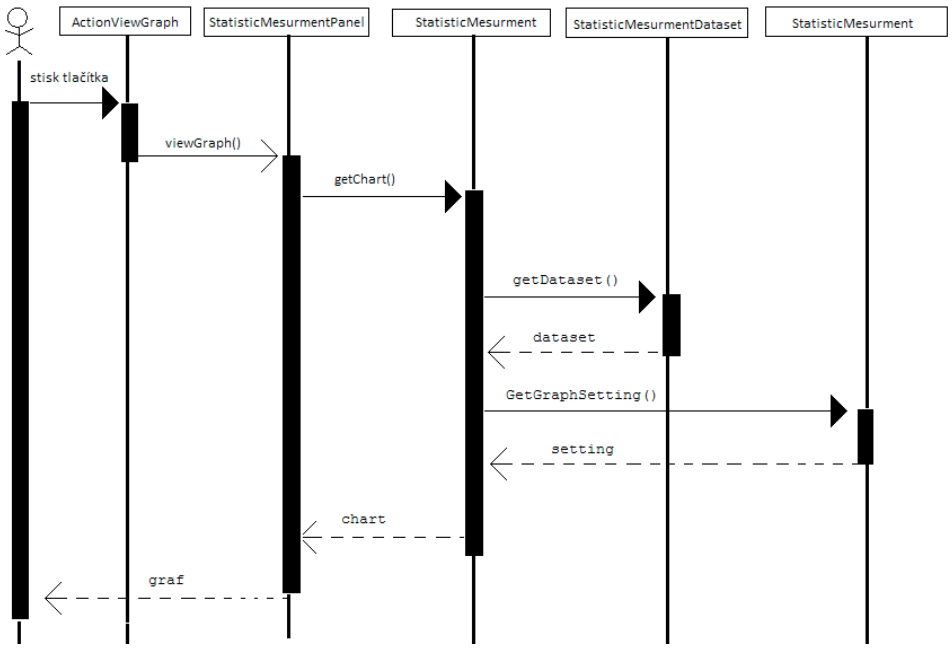
Pomocí výše zmiňovaného tlačítka objektu *Action* je možno vyvolat aplikaci, která naměřené hodnoty a grafy zobrazí v reportu, který se dá uložit v různých formátech (pdf, rtf, Html). Ve vnořené třídě `ActionCreateProtocol` se pak při stisku tlačítka volá procedura *createProtocol*(). Zde se vytvořila instance třídy `WaitOnGetReport`, která zobrazovala informativní zprávu o generování protokolu. Poté se pomocí metody *execute()* spustilo samostatné vlákno s metodou *getReport()* implementovanou ve třídě `StatisticMesurment`, jejíž hlavním úkolem bylo nastavit, zaprvé cestu k šabloně pomocí *GlobalSetting.getInstance().getReporStatTemplate(),* jenž se ukládala do proměnné typu *InputStream* a zadruhé data, která budou následně vypsány v protokolu. Tyto data reprezentuje vnořená třída `StatisticMesurmentReportTableModel`, obsahující pole proměnných a pole datových typů. Důležité je dát si pozor na pořadí zapisovaných prvků, aby každá proměnná měla na stejné pozici v poli přiřazený správný datový typ. Poté byly jednotlivé proměnné v metodě *getValueAt()* roztřízeny podle toho, na jaké místo v dokumentu budou uloženy. Pozici jednotlivých ukládaných hodnot určuje již načtené XML schéma. Tato šablona se nachází v balíčku *resourse*. Pro xy měření se používalo *XYReport.xml*. Tento soubor bylo však potřeba přepracovat, jelikož statistické měření, krom vytváření 2 grafů, také vypočítávalo jiné hodnoty (směrodatná odchylka, minimum, maximum, průměrná hodnota). Soubor byl nazván *reportStatistic.xml.* Úprava toho souboru se řeší v samostatné kapitole 3.3 Úprava souboru reportStatistic. S vytvořením nového souboru se musel upravit i *unisave_setting.xml*, kde byla přidána cesta k nové šabloně. S tím souviselo dopsání nové proměnné *reportstatFileName* a jejich property. Po provedení načtení všech vlastností se daný protokol zobrazil do *protocolFrame*. Celý popis je zobrazen pomocí sekvenčního diagramu (viz obr. 8).



Obrázek 8: Sekvenční diagram tvorby protokolu

Metoda pro vytvoření grafu, histogramu

Podobně jak u *Action* tlačítka na vytvoření protokolu se volá metoda *viewGraph()*. V ní se vytvoří *graphFrame*. Pomocí volané metody *getchart()* implementované v třídě `StatisticMesurment` se nastaví všechny vlastnosti grafu. Podrobnější informace o nastavení jsou popsány v kapitole, věnující se třídě `StatistcMesurment`. Poté se volají metody pro aktualizaci nadpisu grafu a popisu os. A nakonec se vytvoří samotný chart panel, který se zobrazí na obrazovce. U histogramu je tento princip podobný. Rozdíl je ve vlastnostech grafu, jenž je volán jinou metodou.



Obrázek 9: Sekvenční diagram vykreslení grafu

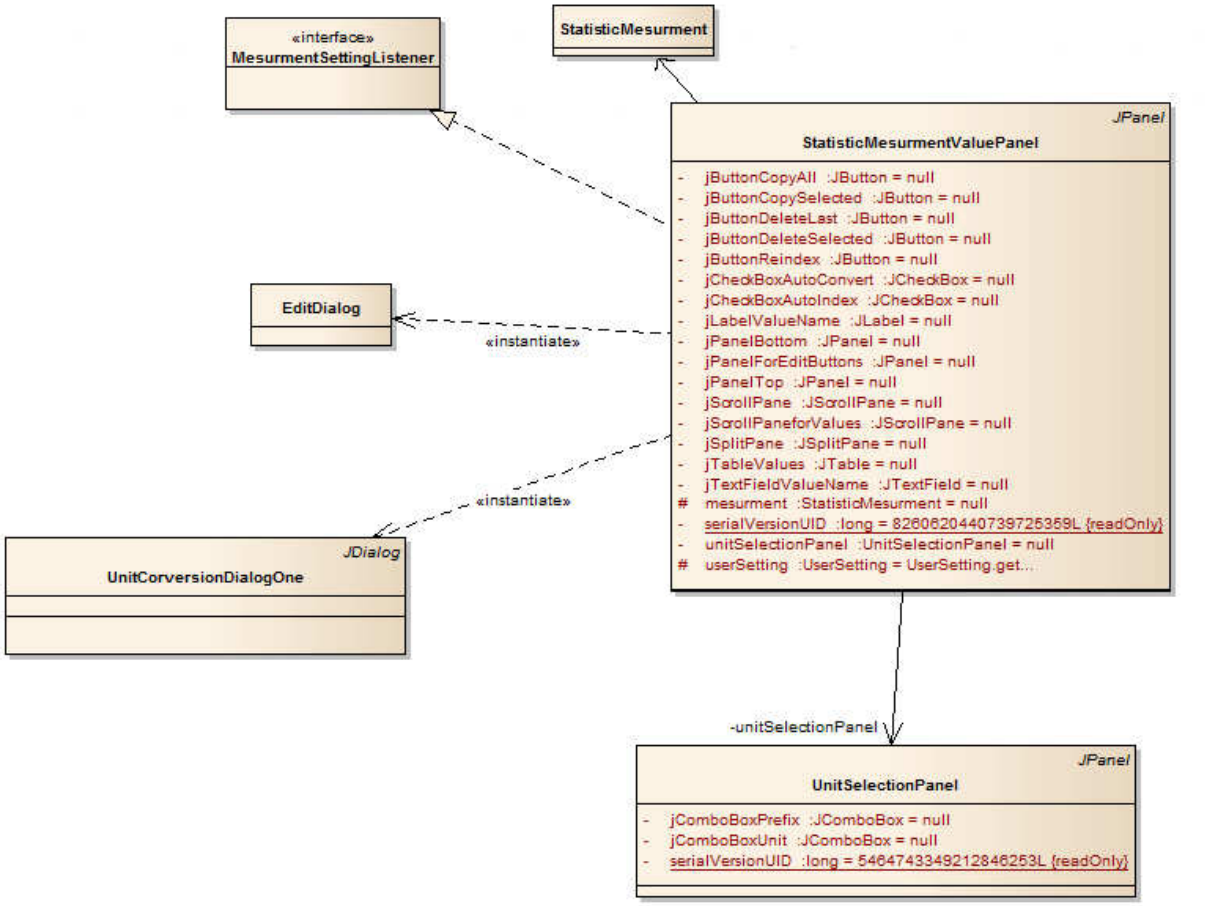
Ostatní metody od ostatních *Action* prvků se vážou na tabulku s naměřenými daty a jsou implementovány ve třídě `StatisticMesurmentPanelValue`.

Poslední větší úprava v této třídě byla v metodě *setMesurment*, ve které se přidali posluchači na jednotlivé akce jako například nastavení grafu nebo dostupnost tlačítek. A pak také samotné nastavení grafu a nastavení formuláře na vytvoření protokolu.

3.1.4 StatisticMesurmentValuePanel

Tato třída reprezentuje zobrazení naměřených hodnot v tabulce, popřípadě její nastavení, editaci (`EditDialog`) nebo konverzi (`UnitConversionDialogOne`). Dále jsou v této třídě doimplementovány metody, které jsou poté volány z `StatisticMesurmentPanel`. Jedná se především o metody reagující na obsah tabulky. Příkladem těchto metody je *Deleteseleced()*, která smaže vybrané záznamy z tabulky, *cutSelectedItems()* nebo *copySelectedItems()* a samozřejmě další metody různého druhu. Tyto metody jsou poté volané objekty *Action*. Jednou z více upravených metod

ve třídě `StatisticMesurmentValuePanel` je *unitConversion()*, která při volání zobrazí nové okno pro změnu jednotek. Toto okno je implementováno ve třídě `UnitConversionDialogOne`, která je podobná třídě `UnitConversionDialog`. Pouze neobsahuje grafické komponenty pro konverzi Y hodnot, které statistické měření nepodporuje.



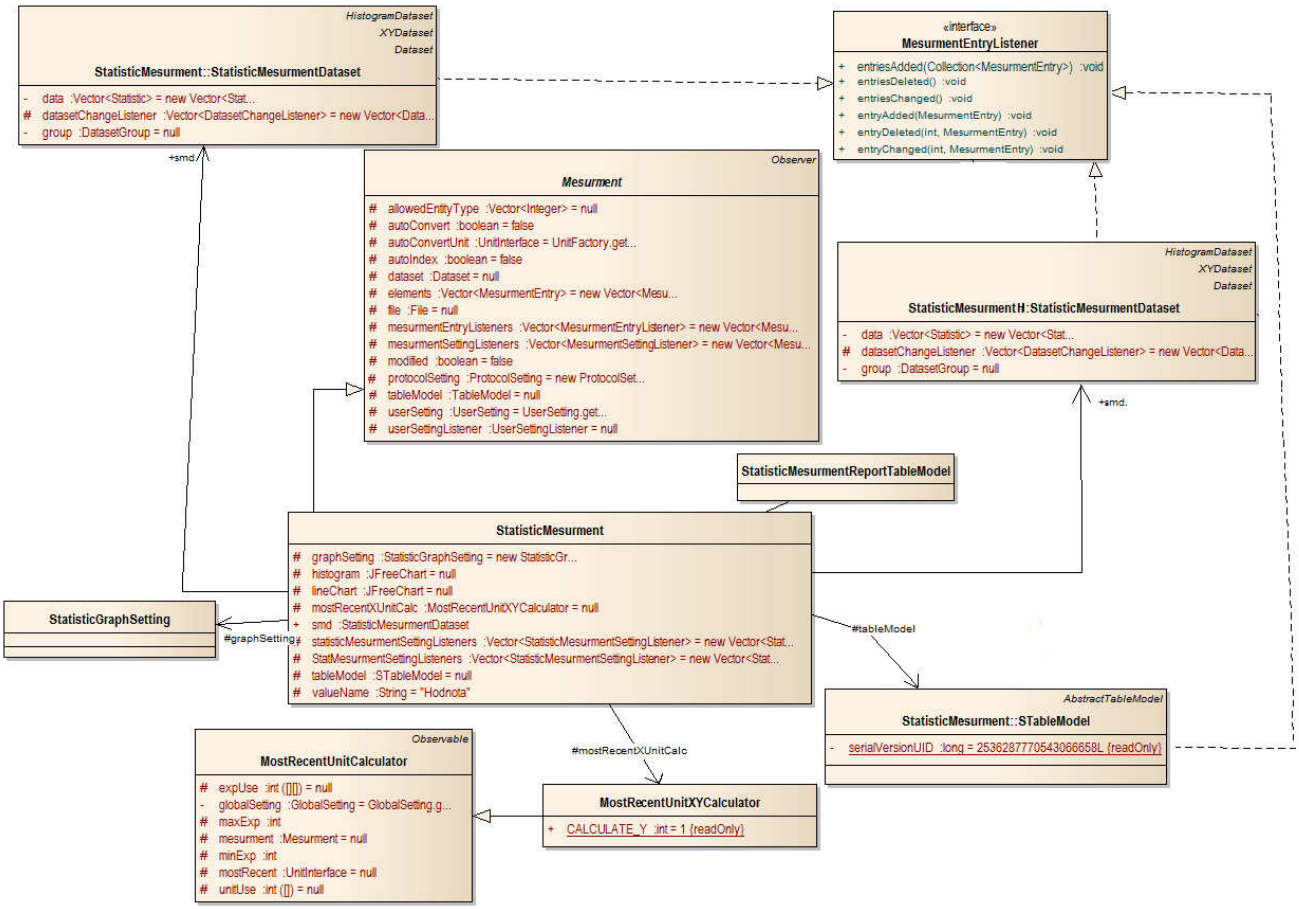
Obrázek 10: UML diagram StatisticMesurmentValuePanel

3.1.5 StatistictValuePanel

Poslední upravenou třídou v v gui balíčku je třída `StatisticValuePanel`. Tato třída se nachází v podbalíčku gui v gui.values a slouží pro vkládání, popřípadě editaci dat. Vytváří ji třída `MesurmentEntitiEditPanelFactory`. Metoda *Inicializace()* vytváří grafické komponenty, které obsahují textové pole pro zadávání hodnot. Samotné tlačítka vložit a stop, jsou implementovány v `ManualGraberFrame`, odkud jsou také volány metody jako *eraseValue()* popřípadě *gainFocus()*. Na výběr jednotky je zde inicializován `UnitSelectionPanel`.

3.2Vytvoření datové části modulu pro statické měření

V balíčku *data* se nachází třída `StatisticMesurment`, jenž si můžeme znázornit v UML diagramu (viz obr. 11). Tato třída dědí ze svého předka `Mesurment` a obsahuje čtyři vnořené třídy `StatisticMesurmentReportTableModel`, `STableModel`, `StatisticMesurmentDataset` a `StatisticMesurmentDatasetH`.



Obrázek 11: UML diagram StatisticMesurment

3.2.1 StatisticMesurmentDataset

Třída `StatisticMesurmentDataSet` slouží jako datový zdroj k tvoření grafů. Obsahuje veškeré metody, jenž jsou důležité pro jejich správné vykreslení, jako je například *getXValue(), getX(), getYValue(), getY()*. V této třídě jsou taktéž přímo implementovány metody pro počítaní statistických hodnot, o kterých byla zmínka v kapitole `StatisticMesurmentPanel`. Jsou to metody *getMin(), getMax, getSmerOdchylka (),getAritPrumer()*. Metody jsou upraveny tak, že při výpočtech jsou vždy naměřené hodnoty převáděny do nejpoužívanějších jednotek. Děje se tak pomocí volání metody *getMostRecentUnit()*. Výhodou takto vytvořeného datasetu je jeho efektivnost, kdy při každém zobrazení grafu se graf znovu netvoří, pouze se načte *dataset*. Ten se sám přepočítává v reakci na změny, které provádíme s naměřenými daty (přidávání hodnot, úprava hodnot, mazaní hodnot). Děje se tak, protože v této třídě jsou implementovány posluchače na změny v tabulce s naměřenými daty, které zavolají metodu *reinsertData()*. Tato metoda smaže předchozí data a znovu načte kolekci upravených měření.

3.2.2 StatisticMesurmentDatasetH

Třída `StatisticMesurmentDataSetH` je obdobou třídy `StatisticMesurmentDataSet`. Používá se jako datový zdroj k vykreslení histogramu. Dědí ze třídy `HistogramDataset`, která implementuje metodu *addSeries(),* jenž je pak volána v metodě *getHset()*. V *getHSet()* se před vložením série musí jednotlivé data přepsat do pole typu *double*. Ostatní metody jsou již stejné jako u třídy `StatisticMesurmentDataSet`.

3.2.3 STableModel

Tato třída obsahuje nastavení samotné tabulky, ve které se zobrazují naměřená nebo vložená data. Jsou zde implementovány například metody *getColumCount(),* udávající počet sloupců nebo *GetValueAt()*, v které se přidávají data jednotlivým sloupcům. Toto rozčlenění se třídí podle druhu měření.

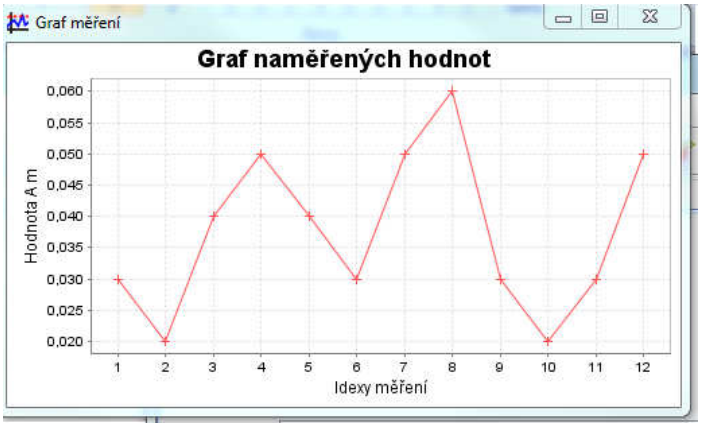
3.2.4 StatisticMesurmentReportTableModel

Jedná se o třídu, která je využívána při tvorbě protokolu. Určuje, která data se mají do protokolu zapsat. Obsahuje 2 pole, jedno obsahuje pole datových typů a druhé názvy proměnných, které jsou pak vepsány v šabloně pro vytvoření protokolu. Metoda *getValueAt()* slouží pro naplnění jednotlivých prvků. Řešena je pomocí příkazu *switch,* který se podle proměnné *columnIndex* naplní. Příkladem může být ukázka, kdy na první pozici v poli COLUM_NAMES na pozici nula je *mesurmentTitle.* Tahle hodnota je získána z těla příkazu *switch* na pozici 0. Na této pozici je volána metoda *protocolSetting.getMesurmentTitle(),* která vrátí nadpis měření (viz kód 15). Takto to probíhá u všech ostatních prvků, které chceme do protokolu zapsat.

```
String[] COLUMN_NAMES = {
                          "mesurmentTitle",
                            . . .
                         }
switch (columnIndex) {
                case 0:
                     ret = protocolSetting.getMesurmentTitle();
```

Kód 15 Ukázka naplnění jednotlivých prvků

Kromě těchto vnořených tříd, se ve `StatistcMesurment` implementují metody *getChart()* a *getChart2()*. Tyto metody jsou typu *JFreeChart*, což je knihovna která umožňuje vykreslovat grafy. Na začátku *getChart()* metody zavoláme *ChartFactory.createXYLineChart(),* pomocí které nastavíme nadpis grafu, popis osy x, popis osy y. Dále získáme zdroj vykreslovaných dat ze třídy `StatisticMesurmentDataset`. Tento zdroj je však potřeba přetypovat na *XYDataset*. Dále získáme námi definované nastavení grafu z *getGraphSetting()* a pozici jak bude graf orientovaný. Pomocí volání dalších metod doimplementujeme barvy grafu, vzhled a barvu čar, textu, popřípadě sloupců, velikost popisků a různá další specifická nastavení. V popiscích os se zobrazují nejvíce používané měrné jednotky, v kterých bylo dané měření prováděno. Výsledný graf pak může vypadat jako na obrázku 12.



Obrázek 12: Vykreslení grafu

Metoda *getChart2(),* která slouží pro vykreslení histogramu, implementuje obdobné metody. *ChartFactory.createHistogram(),* připojuje zdroj dat pomocí metody *getDatasetH(),* jenž vytváří instanci třídy `StatisticMesurmentDatasetH`, v které jsou data přepočítávána.

Kromě dvou metod na vykreslení grafu jsou v této třídě ještě implementovány metody *reidnex()*, jež se používají pro přeindexování prvků vložených v tabulce. Dále jsou zde napsány metody pro vládání dat. Pokud se jedná o vložení pouze jednoho záznamu, volá se metoda *addEntry(),* pokud je záznamů víc je zde připravena metoda *addEntries(),* která má jako vstupní parametr kolekci objektů *MesurmentEntry*.

3.2.5 Úprava souboru reportStatistic

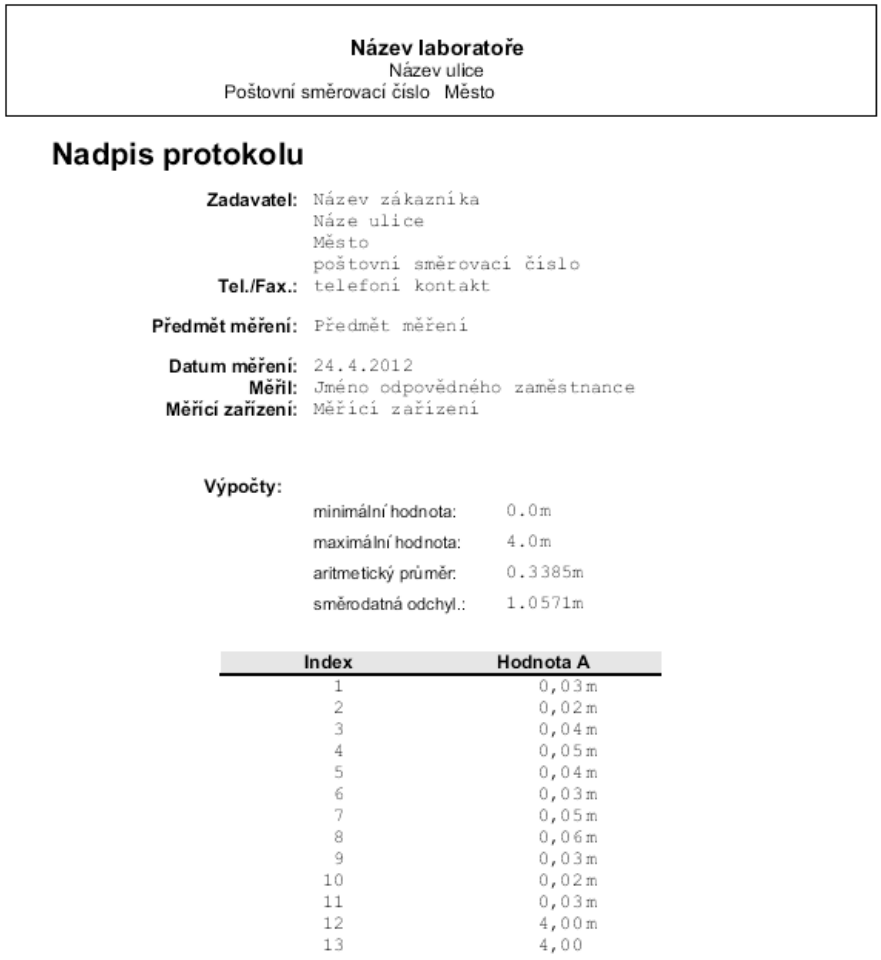
Při tvorbě protokolu se pomocí metod třídy `JFreeReport` načítá šablona pro zobrazení protokolu, jenž je uložena v balíčku *resourse.* U modulu pro XY měření se využíval soubor se jménem *reportXY*, jenž nastavuje grafickou podobu jednotlivých vkládajících se prvků.

U statistického měření bylo potřeba některé prvky smazat nebo naopak vytvořit nové. Takto nově vytvořená šablona *reportStatistic.xml* obsahuje kromě původní hlavičky obsahující název a adresu laboratoře, také základní popis měření například kdo prováděl dané měření, na jakém měřícím zařízení, datum měření atd. Nově vytvořenou částí je úsek statistických výpočtů, které zaznamenávají vypočtené hodnoty z naměřených dat (obrázek 12). Z obrázku lze vyčíst, že oblast statických výpočtů se skládá z popisku (label) a polí, do kterých se vkládají údaje (string-field). X a Y hodnoty určují pozici daného elementu na stránce, width a height zase jeho velikost. V elementu *string-field* v položce *fieldname* je přesně daný název proměnné, kterou na danou pozici chceme vložit.

```
<label x="40"  y="304" width="140"  height="14" fontname="Arial"
       fsbold="true" alignment="right">Výpočty:</label>

<label  x="200"  y="320" width="140"  height="14" fontname="Arial"
       fontsize="11"  alignment="left">minimální hodnota:</label>

<string-field x="325" y="320" width="48" height="14" fontname="Courier New"
       fontsize="12"  fieldname="min" alignment="left"/>

<labelx="200"  y="340" width="140"  height="14" fontname="Arial"
       fontsize="11"  alignment="left">maximální hodnota:</label>

<string-field x="325" y="340" width="48"  height="14" fontname="Courier
       New" fontsize="12"  fieldname="max"  alignment="left"/>
```

Obrázek 13: Ukázka xml šablony

Pod výpočty se nachází tabulka s naměřenými hodnotami. Na druhé straně protokolu se pak nachází graf a histogram. Celý protokol uzavírá pole pro podpis osoby odpovědné za dané měření. Výsledná struktura může vypadat podle obrázku 14.



Název laboratoře
Název ulice
Poštovní směrovací číslo Město

Nadpis protokolu

Zadavatel: Název zákazníka
Náze ulice
Město
poštovní směrovací číslo
Tel./Fax.: telefoní kontakt

Předmět měření: Předmět měření

Datum měření: 24.4.2012
Měřil: Jméno odpovědného zaměstnance
Měřicí zařízení: Měřicí zařízení

Výpočty:

minimální hodnota:	0.0m
maximální hodnota:	4.0m
aritmetický průměr:	0.3385m
směrodatná odchyl.:	1.0571m

Index	Hodnota A
1	0,03m
2	0,02m
3	0,04m
4	0,05m
5	0,04m
6	0,03m
7	0,05m
8	0,06m
9	0,03m
10	0,02m
11	0,03m
12	4,00m
13	4,00

Obrázek 14: Ukázka protokolu

3.Závěr

Jedním z cílů této bakalářské práce bylo upravení načítání z XML souboru pomocí technologie JAXB. Jelikož jsem se s touto technologii setkal poprvé, bylo nejdůležitější si celou problematiku nastudovat, a tím si rozšířit své vědomosti o práci Javy s XML. Poté jsem nově nabité poznatky zkoušel přenést do programu Unisave. Všechny původní XML soubory byly nahrazeny novými, které bylo na začátku potřeba vytvořit. Zde jsem se setkal s několika problémy, které jsem však zdárně vyřešil. U případu načítání celého měření, došlo k situaci, že pokud staré měření (většinou se jedná o měření druhu XY) nebylo uloženo pomocí JAXB, nedalo se znovu načíst. Proto v JAXB výjimce byla ponechána stará technologie načítání. Načtený soubor se pak již uložil pomocí JAXB. Zbylé implementace pomocí SAX jsem okomentoval, popřípadě smazal. Podobně jsem to provedl u vytváření většiny XML souborů pomocí JAXB, aby tyto převody Java objektů na XML soubory zbytečně nezatěžovaly celou aplikaci. Zde právě vidím jednu z možných slabin implementace JAXB, kterou je počet vytváření JAXBcontext , jenž je celkem náročná operace, která může snižovat výkon aplikace. Jediné dva případy, kdy zůstaly implementovány převody v oboru směrech (XML do objektů a naopak) jsou ukládání a načítání uživatelského nastavení a samotné měření. Zjistil jsem, že implementace JaxB technologie je velice snadná, pokud o ní znáte dostatek informací.

Druhou části této bakalářské práce bylo vytvoření statistického modulu. Prvním krokem bylo vytvoření tlačítek pomocí objektů Action, na které později navázaly implementace metod pro vytvoření protokolu. S touto implementací souviselo vytvoření nové šablony a třídy pro vkládání dat na tuto šablonu. Funkcionalita grafu a histogramu byla podmíněna vytvořením datových zdrojů v podobě vnořených tříd, jenž slouží pro jejich vykreslení. Pro počítání statistických výpočtů byly vytvořeny speciální metody a panel, na kterém se zobrazují dané výpočty.

Díky bakalářské práci jsem si rozšířil své znalosti v oblasti *Action* objektů, grafických prvků, vytváření protokolů pomocí JFreeReport, tvorby grafů pomocí JFreeChart a v neposlední řadě také v práci s XML pomocí technologie JAXB. Do budoucna může tato bakalářské práce posloužit jako návod na implementaci dalších typů modulů.

.

Reference

[1] HORTON, Ivor. Java 5. Překlad Petr Poledňák. Praha: Neocortex, 2005, 1443 s. ISBN 80-863-3012-5.

[2] DOMIN, Jan. JAXB – ještě bližší propojení Javy a XML. In: BCVlog [online]. 2011 [cit. 2012-04-30]. Dostupné z: http://blog.bcvsolutions.eu/jaxb-jeste-blizsi-propojeni-javy-a-xml/

[3] Syntaxe XHTML. In: Jakpsatweb [online]. 2004, 2012 [cit. 2012-04-30]. Dostupné z: http://www.jakpsatweb.cz/html/xhtml.html#co

[4] BARANEC, Radovan. Refaktoring a testovanie aplikácie UniSave [online]. Ostrava, 2008 [cit. 2012-04-30]. Dostupné z: http://hdl.handle.net/10084/68228.

[5] Standard Generalized Markup Language. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-04-30]. Dostupné z: http://en.wikipedia.org/wiki/Standard_Generalized_Markup_Language#Standard_versions

[6] Vánoční hrátky s JAXB. In: Youtube [online]. 2010 [cit. 2012-04-30]. Dostupné z: http://www.youtube.com/watch?v=bLfpLcxYuLM

[7] How to Use Actions. In: The Java Tutorials [online]. [cit. 2012-05-01]. Dostupné z: http://docs.oracle.com/javase/tutorial/uiswing/misc/action.html

A Obsah CD

K této práci je přiložené CD – ROM medium, které obsahuje následující soubory:

- Text této bakalářské práce ve formátu PDF
- Zdrojové kódy statistického modulu Unisave (export z Eclipse)

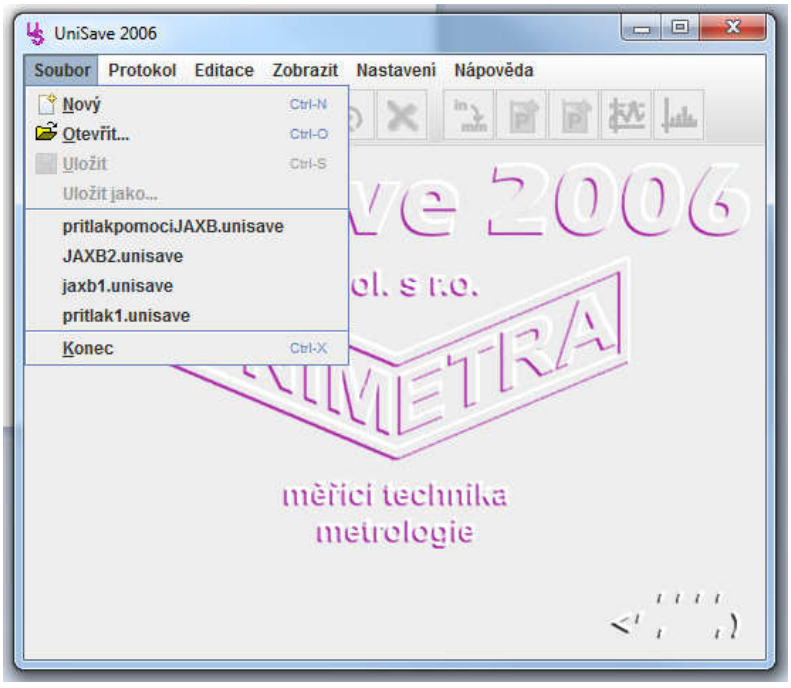
B Uživatelská dokumentace

Úvod

Tato dokumentace popisuje práci s programem Unisave, přesněji s modulem pro statistické měření. Tento modul dokáže z naměřených hodnot (popřípadě ručně zadaných) spočítat statistické údaje jako je maximální, minimální, průměrná hodnota nebo směrodatná odchylka. Z naměřených dat je vytvořen graf a histogram. Všechny tyto data mohou být vygenerovány do protokolu, který je možno uložit do mnoha formátů (pdf, rtf, Html atd). Informace o programu nalezneme v nabídce Nápověda.

1. Úvodní obrazovka

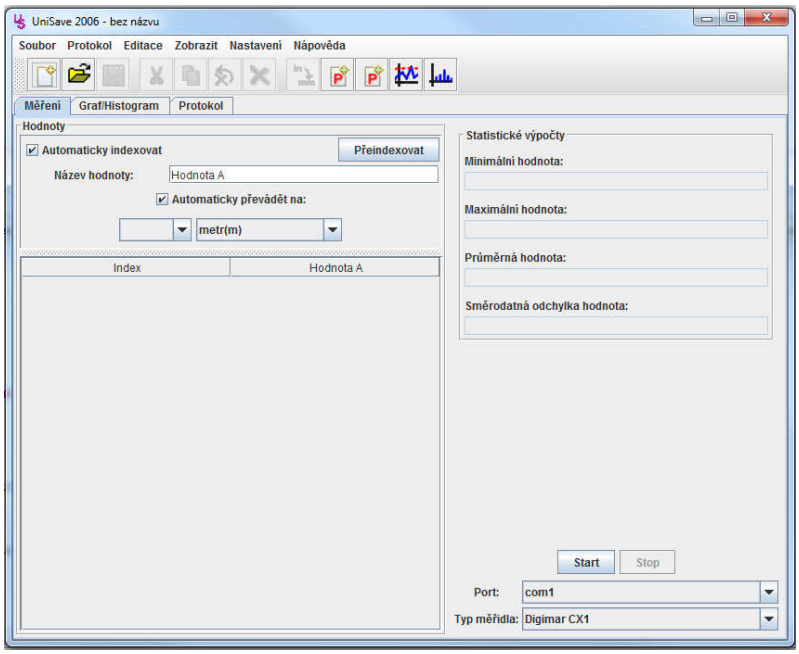
Při samotném spuštění programu se nám objeví úvodní obrazovka. Pomocí tlačítka soubor se nám rozevře nabídka s možnostmi otevření nového měření popřípadě načtení již uloženého měření, ve spodní části této nabídky se nachází naposledy otevřené soubory (viz obrázek 1).



Obrázek 1

2. Nové měření

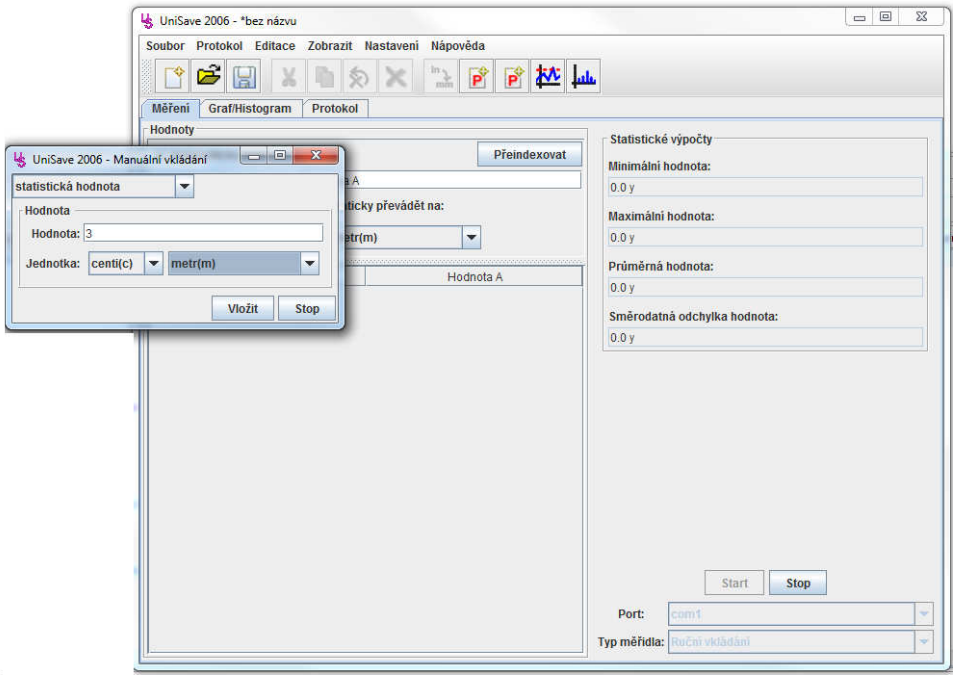
Po stisku tlačítka Nové měření se nám zobrazí panel se záložkami Měření, Graf/Histogram a Protokol. Záložka Měření obsahuje panel pro zobrazení statických výpočtů, dále tabulku naměřených hodnot a nastavení pro načtení hodnot. Pokud budeme chtít zahájit měření, vybereme si port (vlastnosti si můžeme nastavit v nabídce nastavení) popřípadě typ měřidla, z kterého chceme data snímat. Pro manuální vkládání dat zvolíme volbu ruční vkládání a zahájíme měření tlačítkem start (viz obrázek 2).



Obrázek 2

3. Měření hodnot

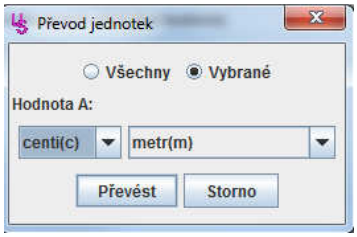
Po stisku tlačítka start se nám zobrazí okno, ve kterém zadáváme naměřené hodnoty. Vybereme zde, v jakých jednotkách se budou data vkládat a pomocí tlačítka Vložit, data vložíme. Pro rychlejší vkládání můžeme využít klávesnicového tlačítka enter. Vkládání opakujeme podle toho, kolik chceme vložit dat. Celé měření ukončíme tlačítkem stop (viz obrázek 3).



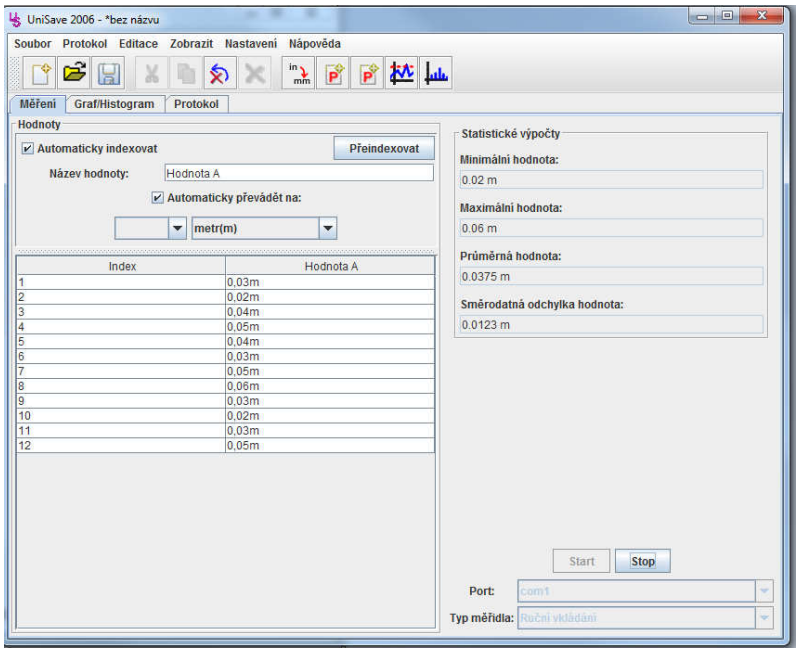
Obrázek 3

4, Po naměření

Po naměření se nám hodnoty přenesly do tabulky (viz obrázek 5). Pokud jsme před měřením měli zaškrtnuté Automaticky převádět, všechna data se nám převedla do jednotek, které jsou zvolené na hlavním panelu (v našem případě se hodnoty převedli z centimetrů na metry). Pokud nám toto převedení nevyhovuje, můžeme pomocí tlačítka Editace zvolit převod jednotek, a hodnoty si převést do námi požadovaných měrných jednotek (viz obrázek 4). Nabídka editace nabízí také další funkce pro úpravu naměřených dat, jako je kopírovaní mazáni, atd. Po úpravě dat se nám někdy mohou pomíchat indexy měření. Proto po zmáčknutí tlačítka přeindexovat se nám indexy opraví. V pravé části okna si můžeme všimnout již vyhodnocení naměřených hodnot, které jsou vždy převáděny do nejvíce používané jednotky v daném měření.



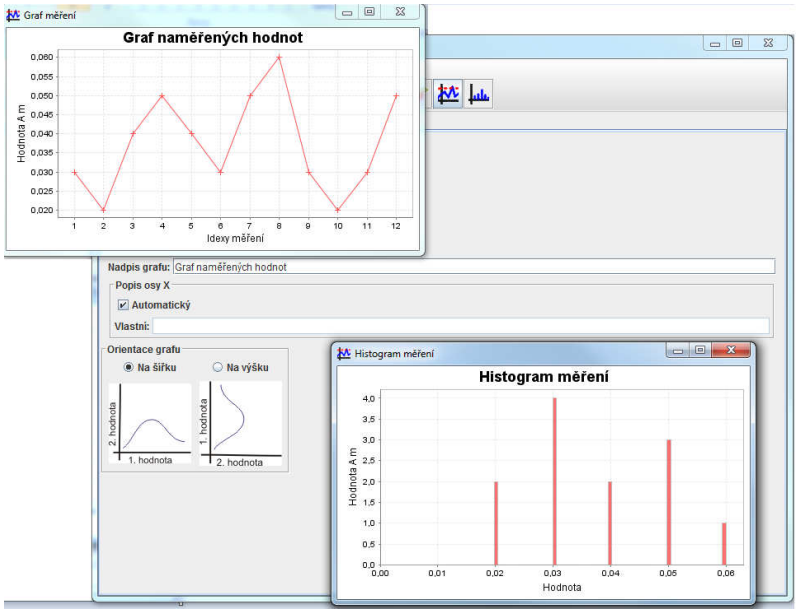
obrázek 4



obrázek 5

5. Záložka Graf/Histogram

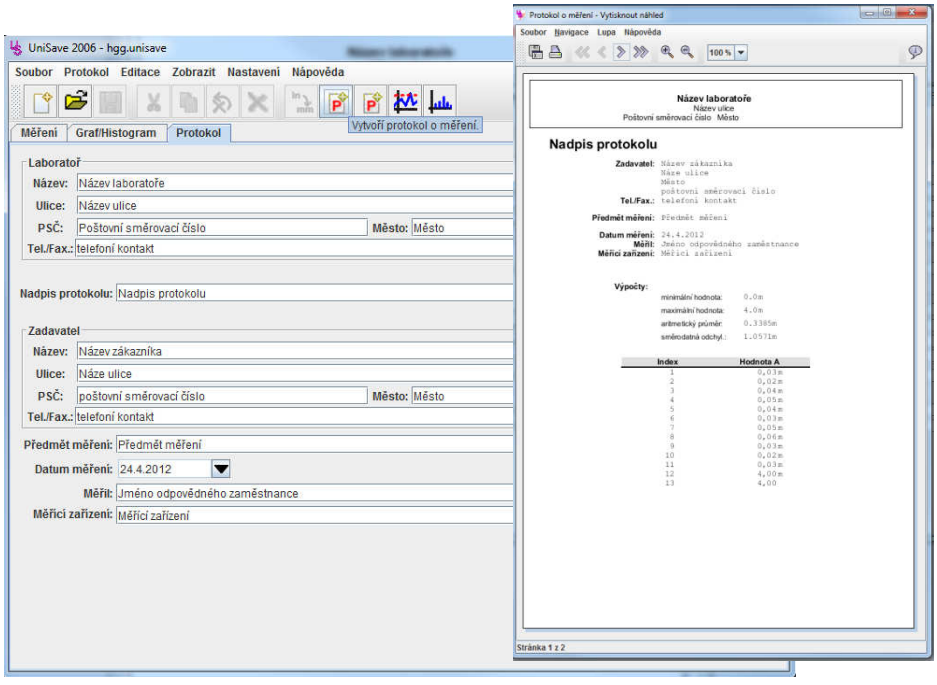
Zde si můžeme předefinovat nastavení grafu (nadpis popřípadě popis osy). Poté se pomocí tlačítka Zobrazit graf popřípadě Histogram objeví naměřené hodnoty (viz obrázek 6).



obrázek 6

6. Záložka Protokol

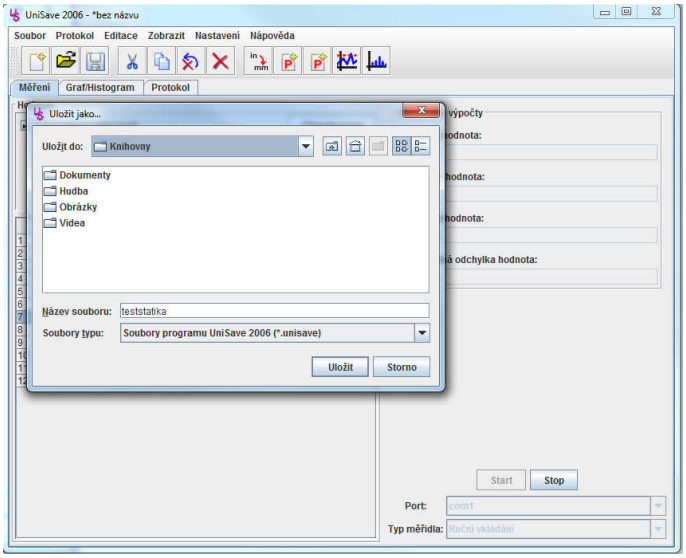
V záložce protokol se nachází formulář pro zadání informací pro vytvoření protokolu. Po vyplnění se může vygenerovat protokol zmáčknutím tlačítka Vytvořit, které se nachází v nabídce Protokol. Takto vygenerovaný protokol si následně může uložit (viz obrázek 7).



obrázek 7

7. Uložení

Nakonec můžeme měření uložit pro pozdější práci či úpravy (viz obrázek 8).



obrázek 8